Improving the Evolutionary Coding for Machine Learning Tasks

Jesús S. Aguilar–Ruiz [1] and **José C. Riquelme**[1] and **Carmelo Del Valle**[1]

Abstract. The most influential factors in the quality of the solutions found by an evolutionary algorithm are a correct coding of the search space and an appropriate evaluation function of the potential solutions. The coding of the search space for the obtaining of decision rules is approached, i.e., the representation of the individuals of the genetic population. Two new methods for encoding discrete and continuous attributes are presented. Our "natural coding" uses one gene per attribute (continuous or discrete) leading to a reduction in the search space. Genetic operators for this approached natural coding are formally described and the reduction of the size of the search space is analysed for several databases from the UCI machine learning repository.

1 INTRODUCTION

In problems related to supervised learning, the decision rules are especially relevant. Given a database with continuous and discrete attributes, and a class label, we try to find a rule set that describes the knowledge within data or classify new unseen data. When the attribute a_i is discrete, the rules take the form of "if $a_i \in \{v_1, \ldots, v_k\}$ then $class$", where the values $\{v_1, \ldots, v_k\}$ are not necessarily all those that the attribute can take. When the attribute a_i is continuous, the rules take the form of "if $a_j \in [l_j, u_j]$ then $class$", where l_j and u_j are two real values belonging to the range of the attribute and $l_j < u_j$. For example, we assume that we have an hypothetical database that associate the weight (in kilograms), height (in meters) and eye color of a person with being or not candidate to have accepted a paper in a relevant conference. The database is a sequence of tuples such as (83, 1.71, green, no), (71, 1.62, black, yes), etc. A rule describing the relationship among attribute values and class might be:

if weight ∈ [60, 70] height ∈ [1.60,1.68] and eye color is blue then he/she is a candidate

The finding of these rules can be tackled with many different techniques and the evolutionary algorithms are among them. Two critical factors have influence on the decision rules obtained by an evolutionary algorithm: the selection of an internal representation of the search space (*coding*) and the definition of an external function that assigns a value of goodness to the potential solutions (*evaluation*).

In this work, we are especially interested in the coding, i.e. in finding a method to accurately encode the genetic information of the individuals. The coding method, named "natural coding" because it uses natural numbers, needs only one value (gene) per attribute, continuous or discrete. In case of continuous attributes, every interval is encoded by one natural number. In case of discrete attributes, one natural number identifies any combination of attribute values. This coding needs a new definition for the genetic operators in order to avoid the convertion from natural numbers to the values of the original space (from genotype to phenotype). These definitions are presented an it is shown in the paper, the evolutionary algorithm can work directly with the natural coding until the end, when the individuals will decoded to decision rules. Our approach leads to a reduction of the search space size, what has a positive influence on the convergence of the evolutionary algorithm.

[1] Department of Computer Science, University of Seville, Spain, e–mail: {aguilar,riquelme,carmelo}@lsi.us.es

2 MOTIVATION

Assuming that the interaction among attributes is not linear, the size of the search space is related to the number of genes used in the coding of the problem. For an individual with size L, the size of the search space is $\prod_{i=1}^{L} |\Omega_i|$, where Ω is the alphabet. Traditionally, the same alphabet has been used for every attribute, mostly the binary alphabet, so that $size = |\Omega|^L$.

To find an adequate encoding to the problem is difficult, at least two recommendations try to be satisfied [10]: coding should have meaningful building blocks and each gene locus should have as few alternatives as possible. The Goldberg's suggestions are especially taken into account in our approach.

Initially, all the studies dedicated to evolutionary algorithms, beginning by the Holland's [12], made use of binary coding. The foundations established by Holland are based on the binary coding and from these every theoretical study have come by the same way. The simplicity and, above all the similarity with the concept of Darwinian analogy, have advanced its theoretical use and practical application.

As regards real coding, theoretically, the main problem is the size of the search space. Since the size of the alphabet is infinite, the number of possible schemes is also infinite and, therefore, many schemes syntactically different but semantically similar will coexist. Many approaches gathered in the bibliography, some of them based on Evolutionary Strategies, use real coding for machine learning tasks [8, 11, 1, 3] or for multiobjective problems [6]. Any value can be coded with an only (real) gene, but this could be avoided using a discrete–real search space, solely allowing the use of certain real values. With this solution the size of the search space is finite and therefore the number of schemes as well.

The main inconvenience of this coding is that it allows genes taking any value of the range, when not all the values would be good as limit of an interval. For example, the C4.5 tool [16] only takes as possible values for the nodes of the decision trees the half–points among two consecutive values of an attribute. This idea is used to code an individual of the genetic population so that only hypothetical good

values will be allowed as conditions over an attribute in the rules.

To clarify this idea, let assume that the next dataset is sorted by the attribute weight.

(57, 1.65, blue, no) (59, 1.60, brown, no)
(60, 1.63, green no) (63, 1.58, brown, yes)
(64, 1.62, black, yes) (68, 1.65, black, no)

In a traditional evolutionary coding for supervised learning based on the Michigan approach [12], a rule is a decoded individual. The left–hand side of the rule is a set of conditions for each attribute: if it is continuous, the condition is formed by two float values; and if it is discrete, the condition is a set of boolean values. Therefore, the interval obtained for the weight would be defined by two values [l_i, u_i], where l_i and u_i are the lower and upper bound of the interval, respectively, $l_i \leq u_i$, and $L_i \leq l_i$ and $u_i \leq U_i$, where L_i and U_i are the lower and upper bounds of the range of the attribute.

C4.5 will investigate as possible values: 57, 59.5, 61.5, 63.5, 66, etc. In other words C4.5 is applying an unsupervised method of discretization since the class label is not taking into account in this process. The reduction of the number of values is only determined by the number of equal values for the attribute being considered. This unsupervised discretization is not a good choice to analyze possible limit values (either using entropy or any other criterion) for the intervals.

A number of remarkable supervised discretization methods has been approached in the bibliography. Among them, the Holte's 1R [13] and the method of Fayyad and Irani [9] were compared to the one used in this work. Our supervised discretization method, named USD (Unparametrized Supervised Discretization), is very similar to 1R, although it does not need any input parameter. USD is based on projections of attribute values [2]. However, as the aim of this method is not to find intervals but cut–points to be used as limits of the further decision rules, we assume that any supervised discretization method would be appropriate for our purpose. As we will see below, if the discretization method produces k intervals, then there will be $k+1$ cutpoints and will therefore be $\frac{k(k+1)}{2}$ possible intervals for the decision rules.

Our goal consists in observing the class along the discretization method and decreasing the alphabet size. Following the example above, we can observe that it is necessary investigate only the values 57, 61.5 and 66, because they are values which produce a change of class. Therefore, this coding allows to use all the possible intervals defined by every two cutpoints obtained by means of discretization.

3 NATURAL CODING

In this section we propose a new coding together with their genetic operators, which will be presented in two independent subsections: discrete and continuous attributes, respectively. This coding has been named "natural" because it only uses natural numbers to represent the set of values for continuous and discrete attributes which can take part of the decision rules.

Through the text, we will use a very simple database in order to explain the application of the genetic operators. This database has one discrete attribute with values {red,green,blue} and a continuous attribute with range [1.1,6.2].

3.1 DISCRETE ATTRIBUTES

Several systems exist that learn concepts (*concept learners*) that use binary coding for discrete attributes: GABIL [7] and GIL [14] are two of the most known. When the set of attribute values has many different values, the length of the individual is very high so that a reduction of the length might influence positively on speeding up the algorithm.

From now on, we will consider that a discrete attribute is coded by an only natural number (one gene) and we will analyse how to apply the crossover and mutation operators in order to the new values retain the meaning that they would have had with the binary coding. The new value will belong to the interval $[0, 2^{|A|}-1]$, where $|A|$ is the number of different values of the attribute. With the natural coding for discrete attributes a reduction of the size of the search space is not obtained, but it is guaranteed that the length of the individual does not take part in the computational cost associated with the genetic operators.

3.1.1 *Natural mutation*

Following the database example, when a value is selected for mutation, for example 3 (011) = {green, blue}, there are three options: 111, 001 and 010, equivalent to the numbers 7, 1 and 2, respectively.

Firstly, we assign the natural number corresponding to each binary number. The mutation of each value would have to be some of the values shown in Table 1. The values appear in rows, from less to more significant, from top to bottom, respectively.

Table 1. Values obtained after mutation.

	0	1	2	3	4	5	6	7
−signifi cant	1	0	3	2	5	4	7	6
	2	3	0	1	6	7	4	5
+signifi cant	4	5	6	7	0	1	2	3

For example, from 0 the values 1, 2 or 4 could be obtained.

Definition 1 (natural mutation) *Let n be the value of a gene of an individual, then the natural mutation of the k^{th} bit of n, denoted by $mut_k(n)$, is the natural number produced by changing that k^{th} bit.*

$$mut_k(n) = \left(n + 2^{k-1}\right) \%2^k + 2^k \left\lfloor \frac{n}{2^k} \right\rfloor \quad (1)$$

where $k \in \{1, 2, \ldots, |A|\}$; $|A|$ is the number of values of the attribute; $mut_k(n)$ are the possible mutated values from n; % is the rest of the integer division; and $\lfloor \cdot \rfloor$ is the integer part.

Example 1 *For example, according to Table 1, the possible values for $n = 5$(101) will be 4, 7 y 1.*

$$mut_1(5) = (5 + 2^0)\%2^1 + 2^1 \left\lfloor \tfrac{5}{2^1} \right\rfloor = 4 \textit{ (100)}$$

$$mut_2(5) = (5 + 2^1)\%2^2 + 2^2 \left\lfloor \tfrac{5}{2^2} \right\rfloor = 7 \textit{ (111)}$$

$$mut_3(5) = (5 + 2^2)\%2^3 + 2^3 \left\lfloor \tfrac{5}{2^3} \right\rfloor = 1 \textit{ (001)}$$

Definition 2 (mutation set) *Let n be the value of a gene, we define mutation set, $Mut(n)$, as the set of all valid mutations for a natural value.*

$$Mut(n) = \bigcup_{k=1}^{|A|} mut_k(n) \quad (2)$$

where $mut_k(n)$ is the natural mutation of the k^{th} binary bit.

Example 2 *From Example 1, $Mut(5) = \{4, 7, 1\}$.*

Note that every discrete attribute value set represented by a natural number can be mutated by using Equation 2, generating another natural number which represents a new discrete attribute value set. Therefore, binary values does not appear in the population. As it is shown in Example 2, from the value 5 ({red,blue}), the possible values are 4 ({red}), 7 ({red,green,blue}) and 1 ({blue}).

3.1.2 Natural crossover

Definition 3 (order–j mutation class) *Let Z be a set of natural numbers, let us define the order–j mutation class, and it will be denoted as $[Mut(Z)]^j$, as follows:*

$$\begin{aligned} [Mut(Z)]^0 &= Z \\ [Mut(Z)]^1 &= \bigcup_{z \in Z} Mut(n) \\ &\vdots \\ [Mut(Z)]^j &= Mut\left([Mut(Z)]^{j-1}\right) \end{aligned} \quad (3)$$

Example 3 *From Table 1, let Z=$\{1\}$, then the order–0, order–1, order–2 and order–3 mutation class of Z will be,*

Order–0: $[Mut(\{1\})]^0 = \{1\}$
Order–1: $[Mut(\{1\})]^1 = \{1, 0, 3, 5\}$
Order–2:
$[Mut(\{1\})]^2 = Mut(1) \cup Mut(0) \cup Mut(3) \cup Mut(5)$
therefore, $[Mut(\{1\})]^2 = \{1, 0, 3, 5, 2, 4, 7\}$
Order–3: $[Mut(\{1\})]^3 = \{1, 0, 3, 5, 2, 4, 7, 6\}$

Definition 4 (natural crossover) *Let n_i and n_j be the values of two genes from two individuals x_i and x_j for the same attribute. The gene of the offspring, $Cross(n_i, n_j)$, will be obtained from the values belonging to the first non–empty intersection among the mutation classes. Let $t \geq 0$, and $Q^t = [Mut(Mut(n_i))]^t \cap [Mut(Mut(n_j))]^t$, then*

$$Cross(n_i, n_j) = \left\{ z \in Q^t \mid Q^t \neq \emptyset, \forall s \geq 0, s < t, Q^s = \emptyset \right\} \quad (4)$$

Example 4 *Let assume that we have the values 0(000) and 3(011). We include the current value into the mutation set since the offspring could be similar to the parents. Thus, 0 mutates to $\{0, 1, 2, 4\}$ and 3 mutates to $\{3, 2, 1, 7\}$. Both genes share 1(001) and 2(010), then any of them could be the offspring from 0 and 3. It is possible that the intersection is the parents (for example, for 1 and 3, as the schema is the same for both, 0*1).*

Example 5 *Let assume that we have the worst case, the values 0(000) and 7(111), which have no bits in common.*

$n_i = 0 \rightarrow Mut(0) = \{0, 1, 2, 4\}$
$n_j = 7 \rightarrow Mut(7) = \{7, 6, 5, 3\}$
$[Mut(Mut(0))]^0 = \{0, 1, 2, 4\}$
$[Mut(Mut(7))]^0 = \{7, 6, 5, 3\}$
Then, $Q^0 = \emptyset$
$[Mut(Mut(0))]^1 = \{0, 1, 2, 4, 3, 5, 6\}$
$[Mut(Mut(7))]^1 = \{7, 6, 5, 3, 4, 2, 1\}$
Therefore, $Q^1 \neq \emptyset$

The intersection will be $\{1, 2, 3, 4, 5, 6\}$, and any of them will be the valid offspring for $Cross(0, 7)$.

3.2 CONTINUOUS ATTRIBUTES

Using binary encoding in continuous domains requires transformations from binary to real for every attribute in order to apply the evaluation function. Moreover, when we convert binary into real, the precision is being lost, so that we have to find the exact number of bits to eliminate the difference between any two values of an attribute. This ensures that a mutation of the less significant bit of an attribute should include or exclude at least one example from the training set. Nevertheless, the real coding is more appropriate with real domains, simply because is more natural to the domain. A number of authors have investigated non–binary evolutionary algorithms theoretically [4, 15]. Two float values would be needed to express the interval of a continuous attribute. For example, for a database with two attributes, continuous and discrete ones, an individual of the population could be as that that is depicted in Figure 1.

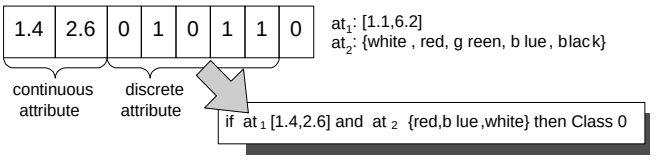

Figure 1. Example of coding.

In case of continuous attributes, the coding is less accurate since the number of values that the attribute can take in its real range is infinite. The search space is therefore larger, so that reducing this search space would make possible a faster convergence of the algorithm . Then, the first step consists in trying to diminish the cardinality of the set of values of the attribute.

3.2.1 Diminishing the cardinality

Firstly, we will analyse what intervals inside the range of the attribute tends to appear as intervals for a possible decision rule obtained from the natural coding. As mentioned before, this task could be solve by any supervised discretization algorithm, for example, the well–known method $1R$ proposed in [13]. Once the vector indicating which are the boundaries for the intervals is obtained (*vector of cuts*), we assign natural numbers to any possible combination, as it appears in Table 2.

Example 6 *Let assume from the database example that the output of the discretization algorithm is the vector of cuts $\{1.1, 3.75, 4.85, 5.2, 6.2\}$. The possible intervals to be generated from those values are shown in Table 2. Each interval is identified by a natural number, for example, the interval $[3.75, 4.85]$ will be referenced by the natural number 6.*

Table 2 shows 5 cutpoints, which can generate 10 intervals. The number of intervals defines the size of the alphabet for such attribute and depends on the number of cuts k, exactly $|\Omega| = \frac{k(k-1)}{2}$. To know the maximum number of cuts that an attribute should have in order to consider interesting this coding is $k < \sqrt{3n}$, where n is the number of different values.

In Table 2 a natural number (in bold), beginning by 1, from left to right and from top to bottom, is assigned to each interval. These "natural" numbers will help us to identify such intervals later.

3.2.2 Transitions

Once the necessary number of cuts has been calculated, we know the size of the new alphabet $|\Omega|$. From now, we will analyse the mutation and crossover operators for this coding.

Definition 5 (row and column) *Let n be the value of the gene, and let r and c be the row and the column, respectively, where n is located in Table 2. The way in which r and c are calculated is: (% is the remainder of the integer division)*

$$r = \frac{n-1}{k-1} + 1 \qquad c = (n-1)\%(k-1) + 1 \quad (5)$$

Table 2. Intervals calculated for the continuous attribute with range $[1.1, 6.2]$.

Cutpoints	3.75	4.85	5.2	6.2
1.1	$\mathbf{1} \equiv [1.1, 3.75]$	$\mathbf{2} \equiv [1.1, 4.85]$	$\mathbf{3} \equiv [1.1, 5.2]$	$\mathbf{4} \equiv [1.1, 6.2]$
3.75	-	$\mathbf{6} \equiv [3.75, 4.85]$	$\mathbf{7} \equiv [3.75, 5.2]$	$\mathbf{8} \equiv [3.75, 6.2]$
4.85	-	-	$\mathbf{11} \equiv [4.85, 5.2]$	$\mathbf{12} \equiv [4.85, 6.2]$
5.2	-	-	-	$\mathbf{16} \equiv [5.2, 6.2]$

Example 7 *Let $n_i = 2$ and $n_j = 8$. Then*

$$r(2) = 1 \quad c(2) = 2 \quad r(8) = 2 \quad c(8) = 4$$

That is to say, 2 is in row 1 and column 2, and 8 is in row 2 and column 4.

Definition 6 (boundaries) *Let n be the value of the gene, we named boundaries of the value n to those values from Table 2 that limits the four possible shifts (one by direction): left, right, up and down, and they will be denoted as ble, bri, bup and bdo, respectively, and they will be calculated as:*

$$\begin{array}{l} ble(n) = (\mathsf{k}-1)(r-1) + r \\ bri(n) = (\mathsf{k}-1)r \\ bup(n) = c \\ bdo(n) = (\mathsf{k}-1)(c-1) + c \end{array} \tag{6}$$

Example 8 *From Example 7,*

$ble(2) = 1 \quad bri(2) = 4 \quad bup(2) = 2 \quad bdo(2) = 6$
$ble(8) = 6 \quad bri(8) = 8 \quad bup(8) = 4 \quad bdo(8) = 16$

That is to say, 2 could reach up to 1 to the left, up to 4 to the right, up to 2 to the top and up to 6 to the bottom; 8 could reach up to 6 to the left, up to 8 to the right, up to 4 to the top and up to 16 to the bottom.

Definition 7 (shifts) *The left, right, up and down adjacent shifts for a value n will be obtained (if possible) as follows:*

$$\begin{array}{l} left(n) = max(ble(n), n-1) \\ right(n) = min(bri(n), n+1) \\ up(n) = max(bup(n), n - \mathsf{k} + 1) \\ down(n) = min(bdo(n), n + \mathsf{k} - 1) \end{array} \tag{7}$$

We define $horizontal$ and $vertical$ shifts as all the possible shifts as for row as for column, respectively, where n is placed, including n.

$$\overline{hor}(n) = \bigcup_{i=1}^{\mathsf{k}-1} max(ble(n), (\mathsf{k}-1)(r-1) + i) \tag{8}$$

$$\overline{ver}(n) = \bigcup_{i=1}^{\mathsf{k}-1} min(bdo(n), (\mathsf{k}-1)(i-1) + c) \tag{9}$$

Example 9 *From Example 8, the adjacent shifts of 2 and 8 will be:*

left(2)=1 right(2)=3 up(2)=2 down(2)=6
left(8)=7 right(8)=8 up(8)=4 down(8)=12
$\overline{hor}(2) = \{1, 2, 3, 4\} \quad \overline{ver}(2) = \{2, 6\}$
$\overline{hor}(8) = \{6, 7, 8\} \quad \overline{ver}(8) = \{4, 8, 12, 16\}$

3.2.3 Natural mutation

A mutation consists in selecting a near interval to that that has the value n. For example, observing Table 2, if the number of cuts is equal to 5 ($\mathsf{k} = 5$), and $n = 7$, there are four possible mutations {3,6,8,11}; however, if $n = 4$, there will be two possible mutations {3,8}.

Definition 8 (natural mutation) *Let n be the value of the gene, the natural mutation of n, denoted by $Mut(n)$, is any near value to n by using the shifts and distinct from n.*

$$Mut(n) \in \{x \mid x \in \{mov(n) - n\}\} \tag{10}$$

where $mov(n) = left(n) \cup right(n) \cup up(n) \cup down(n)$.

Example 10 *Thus, $Mut(2) \in \{\{1, 2, 3, 6\} - \{2\}\}$, i.e., $Mut(2) \in \{1, 3, 6\}$. Now, one of the three values is selected.*

3.2.4 Natural crossover

Definition 9 (natural crossover) *The natural crossover between two values n_i and n_j, denoted by $Cross(n_i, n_j)$ is obtained as follows:*

$$\begin{array}{l} Cross(n_i, n_j) \in \\ \quad \in \left(\left(\overline{hor}(n_i) \cap \overline{ver}(n_j)\right) \cup \left(\overline{hor}(n_j) \cap \overline{ver}(n_i)\right)\right) \end{array} \tag{11}$$

We can observe in Table 2 that the nearest values are in the intersection between the row and the column where both values being crossed are placed. For example, the nearest value to 1 and 6 is 2; the nearest value to 6 and 12 is 8. Only when the values n_i and n_j are located in the same row or column the interval will be inside the other.

Example 11 *Thus,*
$Cross(2, 8) \in$
$\quad \in \{\{\{1, 2, 3, 4\} \cap \{4, 8, 12, 16\}\} \cup \{\{6, 7, 8\} \cap \{2, 6\}\}\}$,
i.e., $Cross(2, 8) \in \{4, 6\}$.
Now, we can select one or more of the values generated by the crossover operator.

4 DISCUSSION

For the database example, a possible individual of the genetic population is shown in Figure 2. This individual information means *if a_1 is green or blue and a_2 is in $[3.75, 5.2]$ then class=1*.

In Table 3 are shown the databases from the UCI repository [5] used in the analysis. In case of continuous attributes, to calculate the number of genes (length L) for the binary coding Equation 12 was used, where l_i and u_i are the lower and upper values of the domain and δ is the smaller difference among any two values of the attribute.

$$L = \left\lceil \lg_2 \left(1 + \frac{u_i - l_i}{\delta}\right) \right\rceil \tag{12}$$

Discrete attributes were encoded using a single gene for each value. In the column labelled "Natural" the genes for binary coding which should be used are indicated as the number of attributes of the database. It is noteworthy that the length of individuals has been considerably decreased. Nevertheless, as we will se below, the most important fact is the decreasing of the size of the search space.

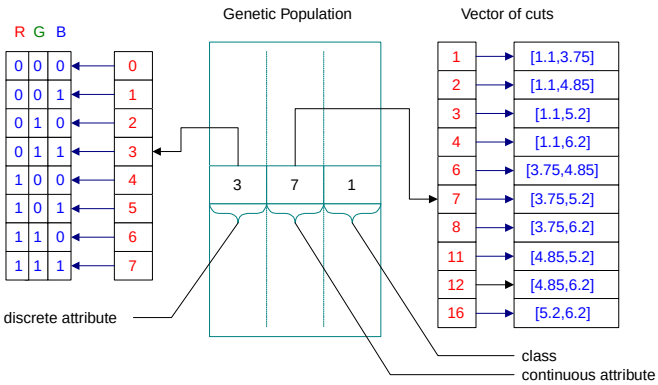

Figure 2. Example: discrete and continuous attributes.

Table 3. Length: length of individuals of the genetic population for binary and natural coding. Search Space Size: size of the search space for both codings, indicated as base–10 logarithms.

	Length		Search Space Size (10^m)	
Batabase	Binary	Natural	Binary(m)	Natural(m)
Autos	366	25	110.2	46.3
Balance	12	4	3.6	3.1
Breat Cancer-W	87	9	26.2	7.1
Cleveland Rating	202	15	60.8	16.3
German Credit	95	20	28.6	25.1
Glass	86	9	25.9	16.4
Hepatitis	264	19	79.5	15.0
Ionosphere	551	34	165.9	75.8
Iris	22	4	6.6	3.7
Led7	14	7	4.2	4.2
Letter	256	16	70.1	30.9
Monk1	17	6	5.1	5.1
Mushroom	125	22	37.6	37.6
Pima	65	8	19.6	18.4
Sonar	757	60	227.9	136.7
Soybean	100	35	30.1	10.5
Tic-Tac-Toe	27	9	8.1	8.1
Vote	32	16	9.6	9.6
Wine	120	13	36.1	23.8
Zoo	134	17	10.8	10.8

As for this last aspect, indicated as logarithms to the base 10, in column "Binary(m)" appears the number of possible solutions into the space for the binary coding, which have been obtained from the length of the individuals. In column "Natural(m)" appears the figure calculated from the product of all possible combinations of values for every attribute (discrete values and different intervals for continuous attributes). In some cases the cardinality of the search space is similar for both codings due to the absence of continuous attributes in the database. The improvement of the natural coding with respect to the binary coding can be observed by simply subtracting exponents.

5 CONCLUSIONS

In this paper a new coding method for evolutionary algorithms in supervised learning is presented. This method converts every attribute domain into a natural number domain. The population will therefore have only natural numbers. The genetic operators (mutation and crossover) are defined in order to work efficiently with this new search space, and they use no convertions from the original attribute domains to the natural number domains, but the evolutionary algorithm works from the begining to the end with natural numbers.

As every attribute, continuous or discrete, use one gene in the individual of the population, the size of the search space is decreased, what might allow a faster convergence of the evolutionary algorithm.

6 FUTURE WORKS

Currently, the ongoing experiments show that this natural coding obtains better results (error rate and number of rules) than binary versions of the evolutionary algorithm. However, the most important feature is that as a consecuence of the smaller search space size the computational cost is approximately half.

ACKNOWLEDGEMENTS

The authors are grateful to David Fogel for their suggestions and to Erick Cantú–Paz for helpful comments. Special thanks to William Langdon for their constructive criticisms and very valuable discussion. The research was supported by the Spanish Research Agency CICYT under grant TIC1143–C03–02.

REFERENCES

[1] J. S. Aguilar-Ruiz, J. C. Riquelme, and M. Toro, 'A tool to obtain a hierarchical qualitative set of rules from quantitative data', in *Lecture Notes in Artifi cial Intelligence 1415. Springer-Verlag*, pp. 336–346, (1998).

[2] J. S. Aguilar-Ruiz, J. C. Riquelme, and M. Toro, 'Data set editing by ordered projection', in *Proceedings of the* 14^{th} *European Conference on Artifi cial Intelligence (ECAI'00)*, pp. 251–255, Berlin, Germany, (August 2000).

[3] J. S. Aguilar-Ruiz, J.C. Riquelme, and M. Toro, 'Three geometric approaches for representing decision rules in a supervised learning system', in *Genetic and Evolutionary Computation Conference (GECCO '99)*, p. 771, Orlando, Florida, EE.UU., (1999).

[4] S. Bhattacharyya and G.J. Koehler, 'An analysis of non–binary genetic algorithms with cardinality 2^v', *Complex Systems*, **8**, 227–256, (1994).

[5] C. Blake and E. K. Merz. UCI repository of machine learning databases, 1998.

[6] Kalyanmoy Deb and Amarendra Kumar, 'Real-coded Genetic Algorithms with Simulated Binary Crossover: Studies on Multimodal and Multiobjective Problems', *Complex Systems*, **9**, 431–454, (1995).

[7] K. A. DeJong, W. M. Spears, and D. F. Gordon, 'Using genetic algorithms for concept learning', *Machine Learning*, **1**(13), 161–188, (1993).

[8] L. J. Eshelman and J. D. Schaffer, 'Real-coded genetic algorithms and interval-schemata', *Foundations of Genetic Algorithms-2*, 187–202, (1993).

[9] U. M. Fayyad and K. B. Irani, 'Multi-interval discretisation of continuous valued attributes for classifi cation learning', in *Proceedings of the Thirteenth International Joint Conference on Artifi cial Intelligence*. Morgan Kaufmann, (1993).

[10] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

[11] F. Herrera, M. Lozano, and J.L. Verdegay, 'Tackling real-coded genetic algorithms: Operators and tools for the behaviour analysis', *Artifi cial Intelligence Review*, **12**, 265–319, (1998).

[12] J. H. Holland, *Adaptation in natural and artifi cial systems*, Ph.D. dissertation, University of Michigan, 1975.

[13] R. C. Holte, 'Very simple classifi cation rules perform well on most commonly used datasets', *Machine learning*, **11**, 63–91, (1993).

[14] C. Z. Janikow, 'A knowledge-intensive genetic algorithm for supervised learning', *Machine Learning*, **1**(13), 169–228, (1993).

[15] G.J. Koehler, S. Bhattacharyya, and M.D. Vose, 'General cardinality genetic algorithms', *Evolutionary Computation*, **5**(4), 439–459, (1998).

[16] J. R. Quinlan, *C4.5: Programs for machine learning*, Morgan Kaufmann, San Mateo, California, 1993.